Cromemco COBOL

ANSI-74 Standard COBOL Features – Complete Level I elements of all modules required by the Standard

NUCLEUS
SEQUENTIAL FILES
TABLE HANDLING

With

INDEXED and **RELATIVE FILES** for flexible applications
COPY statement to speed development by including your source libraries

Powerful Level 2 Features

COMPUTE simplifies computations by combining multiple arithmetic statements
OPEN EXTEND permits you to add records to an existing sequential file without first copying all previous records to a new file
CALL to execute your standard, pre-compiled routines with a single statement
STRING easily combines several data items into one field
UNSTRING easily separates one data item into several fields

Debugging Extensions

EXHIBIT, READY TRACE, RESET TRACE for rapid error isolation

YOUR LOCAL DEALER IS

Cromemco COBOL Features

- Based on American National Standard X3.23-1974
- Relative, Indexed, and Sequential Files
  ACCESS SEQUENTIAL, RANDOM, or **DYNAMIC**
  OPEN EXTEND to add records to a sequential file
  READ NEXT in DYNAMIC ACCESS mode
  START to begin reading a sequential file at a specific record
  READ, WRITE, REWRITE, DELETE records in a file
- Types of Data:

alphanumeric	—character string of ASCII values;
report	—up to 30 digits or special editing characters;
external decimal	—up to 18 digits stored as ASCII, one digit per byte;
internal decimal	—up to 18 digits stored as BCD, two digits per byte;
binary	—two bytes used to store the binary value of an integer in the range -32768 to 32767;

- Tables
  3 levels of subscripting (eg: RATE (I, J, K,))
  INDEXED BY clause in table definition
  SEARCH a table until a specific condition is true
  SEARCH ALL to search a table using a fast, non-serial search method
  Relative Indexing (eg: RATE (I + 3, J—1))
- Qualification of Identifiers
- **COPY** Statement to include a library file of COBOL statements or routines for compilation
- **CALL** and **LINKAGE SECTION** to execute a subprogram
- **COMPUTE** to assign the value of a compound arithmetic expression to one or more variables
- **PERFORM VARYING/UNTIL, AFTER/UNTIL** to execute a subroutine repeatedly until specific conditions are true, automatically changing subscript values before each execution
- Powerful Conditional Features:
  Nested IF's
  AND, OR, NOT in Conditions
  PARENTHESES in Conditions
  SIGN and ZERO test
  Level 88 Condition Names with value series or value range
  Algebraic Relational Symbols $<$, = , $>$
  Implied subject and relation in conditions
- Debugging verbs **EXHIBIT, READY TRACE, RESET TRACE**
- English description of syntax errors

Cromemco
incorporated
Tomorrow's Computers Today
280 BERNARDO AVE. MOUNTAIN VIEW, CA 94043

Part No. 023-0070

July 1979

Cromemco COBOL Elements

NOTATION

< > terms enclosed in "< >" describe operands that occupy those positions in actual use.

[ ] clauses enclosed within brackets are optional. Brackets may be nested.

{ } braces surrounding a list of terms separated by the vertical bar "|" indicate that one of the terms MUST be used. Braces surrounding a clause indicate that the clause may be treated as a unit when followed by the ellipsis (. . .). Braces may be nested.

. . . the ellipsis indicates that the previous item or clause may be repeated an indefinite number of times.

BOLDFACE — words in boldface are required in the clause

UPPER CASE — words in upper case are optional "noise words".

lower case — words in lower case represent generic terms for which the user must substitute valid entries.

COBOL PROGRAM STRUCTURE

```
IDENTIFICATION DIVISION.
PROGRAM-ID.          program-name.
[AUTHOR.             comment-entry . . .]
[INSTALLATION.       comment-entry . . .]
[DATE-WRITTEN.       comment-entry . . .]
[DATE-COMPILED.      comment-entry . . .]
[SECURITY.           comment-entry . . .]

ENVIRONMENT DIVISION.
[CONFIGURATION SECTION.
[SOURCE-COMPUTER.    entry ]
[OBJECT-COMPUTER.    entry ]
[SPECIAL-NAMES.      entry ]  ]
[INPUT-OUTPUT SECTION.
FILE-CONTROL.        entry . . .
[I-O-CONTROL.        entry . . . ]  ]

DATA DIVISION.
[FILE-SECTION.
[file description entry
 record description entry . . . ]  . . . ]
[WORKING-STORAGE SECTION.
[data item description entry . . . ]  . . . ]
[LINKAGE SECTION.
[data item description entry . . . ]  . . . ]

PROCEDURE DIVISION [USING identifier . . . ].
[DECLARATIVES.
{section-name SECTION. USE Sentence.
[paragraph-name. [sentence] . . . ]  . . . } . . .
END DECLARATIVES.]
{ [section-name SECTION. ]
[paragraph-name. [sentence] . . . ]  . . . } . . .
```

FILE-CONTROL Details

SEQUENTIAL FILES:

```
SELECT  file-name ASSIGN [DISK  |  PRINTER}
        [RESERVE integer AREAS ]
        [FILE STATUS IS data-name ]
        [ACCESS MODE IS SEQUENTIAL ]
        [ORGANIZATION IS SEQUENTIAL ].
```

INDEXED AND RELATIVE FILES:

```
SELECT  file-name ASSIGN DISK
        [RESERVE integer AREAS ]
        [FILE STATUS IS data-name ]
        ORGANIZATION IS { INDEXED  |  RELATIVE}
        ACCESS MODE IS {SEQUENTIAL |
                        RANDOM |
                        DYNAMIC }
        { RECORD  |  RELATIVE} KEY IS data-name.
```

Data Item Description Details

```
level-number   {data-name  |  FILLER}
               [REDEFINES data-name ]
               PICTURE size-and-editing-
                       character-string
               [BLANK WHEN ZERO ]
               [JUSTIFIED RIGHT ]
               [  OCCURS integer TIMES
                  [INDEXED BY index-name ]
                  [{ASCENDING  |  DESCENDING}
                   KEY IS data-name ]  ]
               [SIGN IS {TRAILING | LEADING }
                     [SEPARATE CHARACTER ]  ]
               [USAGE IS {COMPUTATIONAL  |
                          INDEX  |
                          COMPUTATIONAL-3 |
                          DISPLAY} ]
               [VALUE IS literal ].
```

Conditions

Simple Conditions:

Relational test:

```
operand [NOT]  <relation> operand
```

Class test:

```
data-name is[NOT]  {NUMERIC  |
                    ALPHABETIC}
```

Sign test:

```
data-name is[NOT]  {NEGATIVE  |
                    ZERO  |
                    POSITIVE}
```

Condition-name test:

```
<level-88 Data Division entry>
```

Compound Conditions:

```
<simple condition>  [ {AND | OR}
                      <simple condition>  ] . . .
```

PROCEDURE DIVISION STATEMENTS

```
ACCEPT  data-name.
ADD  {<numeric-item>  |  literal} . . .
   {TO  |  GIVING} data-name
     [ROUNDED]
     [ON SIZE ERROR  <statements>].
ALTER  paragraph TO PROCEED TO procedure-name.
CLOSE  file-name  [WITH LOCK] . . .  .
COMPUTE  data-name  [ROUNDED] . . . =
        {<numeric-item>  |  literal  |
         <compound arithmetic expression>}
        [ON SIZE ERROR  <statements>].
COPY  source-code-file-name.
DISPLAY  {data-name  |  literal} . . .  .
DIVIDE  {<numeric-item>  |  literal} {BY  |  INTO}
          {<numeric-item>  |  literal}
           [GIVING data-name]  [ROUNDED ]
          [ON SIZE ERROR  <statements>].
EXIT.
GO TO procedure-name . . . [ DEPENDING ON
                                 data-name].
IF  <condition>  <statements>
                    [ELSE  <statements>].
INSPECT  data-name
   [TALLYING data-name FOR
      {CHARACTERS  |
      {ALL  |  LEADING}  <1-char-item>}
      [{BEFORE  |  AFTER}
       INITIAL  <1-char-item> ]  ]
   [REPLACING {CHARACTERS  |
      {ALL  |  LEADING  |  FIRST}  <1-char-item>}
    [{BEFORE  |  AFTER}  INITIAL  <1-char-item>  |  ]
MOVE  {data-name  |  literal } TO data-name . . .  .
MULTIPLY  {<numeric-item>  |  literal}
          BY  {<numeric-item>  |  literal  }
           [GIVING data-name]  [ROUNDED ]
           [ON SIZE ERROR  <statements>].
OPEN  {INPUT  |  I-O  |  OUTPUT  |  EXTEND}
           file-name . . .  .
PERFORM  <range>  [<numeric operand>  TIMES].
PERFORM  <range>  [VARYING data-name FROM
                          amount BY amount]
          UNTIL <condition>.
READ  file-name [INTO data-name].
REWRITE  record-name [FROM data-name].
STOP  {RUN  |  literal}.
STRING  { <alpha-operand> . . . DELIMITED BY
          {<alpha-operand>  |  SIZE}  } . . .
    INTO  data-name  [WITH POINTER data-name]
    [ON OVERFLOW <statement>].
SUBTRACT  {<numeric-item>  |  literal }  . . . FROM
          {<numeric-item>  |  literal }
          [GIVING data-name]  [ROUNDED ]
          [ON SIZE ERROR  <statements>].
UNSTRING  data-name
    [DELIMITED BY [ALL]  <1-char-item>
       [OR [ALL]  <1-char-item>] . . .  ]
    INTO
       {data-name  [DELIMITER IN data-name]
                   [COUNT IN data-name]  }  . . .
    [WITH POINTER data-name]
    [TALLYING IN data-name]
    [ON OVERFLOW  <statement>].
WRITE  record-name  [FROM data-name]
    [{BEFORE  |  AFTER} ADVANCING
    [{<numeric-item>  |  literal}]  {LINES  |  PAGE}].
```

DEBUGGING STATEMENTS

```
EXHIBIT  {literal  |  data-name}.
READY TRACE.
RESET TRACE.
```

INTER-PROGRAM COMMUNICATION

In Calling Program:

```
CALL "<program name>" USING data-name . . .
```

In Called Program:

```
        .
        .
LINKAGE SECTION.
    data-name-description . . .
        .
        .
PROCEDURE DIVISION USING data-name . . .
        .
        .
EXIT PROGRAM.
```

TABLE HANDLING

```
SEARCH  table
        [VARYING  {identifier  |  index-name}]
    [AT END <statement>]
    {WHEN <condition>
        {NEXT SENTENCE  |  <statements>}  }. . .
SEARCH ALL  table
    [AT END <statements>]
    WHEN <simple condition>  {NEXT SENTENCE  |
                                  <statements>}.
    (Note: table must have ASCENDING or DESCENDING
        KEY clause)
SET  {index-name  |  index-item  |  data-name }. . .
        TO {index-name | index-item | data-name |
            integer}.
SET  index-name . . . {UP  |  DOWN}  BY
                  {integer  |  data-name}.
```

INDEXED AND RELATIVE FILES

```
DELETE  file-name RECORD
    [INVALID KEY <statements>].
```

Read Sequential:

```
READ  file-name  [NEXT]  [INTO data-name]
    [AT END <statements>].
```

Read Random or Dynamic:

```
READ  file-name  [INTO data-name]
                 [KEY IS data-name]
    [INVALID KEY <statements>].
REWRITE  record-name  [FROM data-name]
    [INVALID KEY <statements>].
START  file-name  [KEY IS {GREATER  |  NOT LESS  |
                    EQUAL}  data-name]
    [INVALID KEY <statements>].
WRITE  record-name  [FROM data-name]
    [INVALID KEY <statements>].
```

I/O VERBS PERMITTED WITH INDEXED FILES

ACCESS MODE	PROCEDURE DIVISION STATEMENT	OPEN MODE IN EFFECT		
		INPUT	OUTPUT	I-O
SEQUENTIAL	READ	X		X
	WRITE		X	
	REWRITE			X
	START	X		X
	DELETE			X
RANDOM	READ	X		X
	WRITE		X	X
	REWRITE			X
	START			
	DELETE			X
DYNAMIC	READ	X		X
	WRITE		X	X
	REWRITE			X
	START	X		X
	DELETE			X

PERMISSIBLE MOVE OPERANDS

SOURCE OPERAND	RECEIVING OPERAND					
	NUMERIC INTEGER	NUMERIC NON-INTEGER	NUMERIC EDITED	ALPHANUMERIC EDITED	ALPHA-NUMERIC	GROUP
NUMERIC INTEGER	OK	OK	OK	OK (A)	OK (A)	OK (B)
NUMERIC NON-INTEGER	OK	OK	OK			OK (B)
NUMERIC EDITED				OK	OK	OK (B)
ALPHANUMERIC EDITED				OK	OK	OK (B)
ALPHANUMERIC	OK (C)	OK (C)	OK (C)	OK	OK	OK (B)
GROUP	OK (B)	OK (B)	OK (B)	OK (B)	OK (B)	OK (B)

KEY:

(A)— Source sign, if any, is ignored.

(B)— If the source operand or the receiving operand is a Group Item, the move is considered to be a Group MOVE (character by character from left to right with no conversion or editing).

(C)— Source is treated as an unsigned integer, and source length may not exceed 31.

NOTES:

1— Conversion of data to receiving item type occurs for numeric or alphanumeric MOVES to numeric or edited numeric items.

2— No distinction is made in the compiler between alphabetic and alphanumeric; one should not MOVE numeric items to alphabetic items and vice-versa.

3— A numeric item is either external decimal, internal decimal, binary, numeric literal, or Figurative Constant ZERO.

ASCII Character Codes

HEX	CHAR	HEX	CHAR	HEX	CHAR
00	CTRL-@	2B	+	56	V
01	CTRL-A	2C	,	57	W
02	CTRL-B	2D	-	58	X
03	CTRL-C	2E	.	59	Y
04	CTRL-D	2F	/	5A	Z
05	CTRL-E	30	0	5B	[
06	CTRL-F	31	1	5C	\
07	CTRL-G	32	2	5D	]
08	BS	33	3	5E	↑
09	HOR. TAB	34	4	5F	←
0A	LINE FEED	35	5	60	`
0B	VERT. TAB.	36	6	61	a
0C	FF	37	7	62	b
0D	CR	38	8	63	c
0E	CTRL-N	39	9	64	d
0F	CTRL-O	3A	:	65	e
10	CTRL-P	3B	;	66	f
11	CTRL-Q	3C	<	67	g
12	CTRL-R	3D	=	68	h
13	CTRL-S	3E	>	69	i
14	CTRL-T	3F	?	6A	j
15	CTRL-U	40	@	6B	k
16	CTRL-V	41	A	6C	l
17	CTRL-W	42	B	6D	m
18	CTRL-X	43	C	6E	n
19	CTRL-Y	44	D	6F	o
1A	CTRL-Z	45	E	70	p
1B	CTRL-[	46	F	71	q
1C	CTRL-\	47	G	72	r
1D	CTRL-]	48	H	73	s
1E	CTRL-↑	49	I	74	t
1F	CTRL-_	4A	J	75	u
20	SPACE	4B	K	76	v
21	!	4C	L	77	w
22	"	4D	M	78	x
23	#	4E	N	79	y
24	$	4F	O	7A	z
25	%	50	P	7B	{
26	&	51	Q	7C	|
27	'	52	R	7D	}
28	(	53	S	7E	~
29	)	54	T	7F	DEL
2A	*	55	U		

CTRL = Control Character BS = Backspace
CR = Carriage Return FF = Form Feed DEL = Rubout

ASCII Character Codes

DEC	CHAR	DEC	CHAR	DEC	CHAR
000	CTRL-@	043	+	086	V
001	CTRL-A	044	,	087	W
002	CTRL-B	045	-	088	X
003	CTRL-C	046	.	089	Y
004	CTRL-D	047	/	090	Z
005	CTRL-E	048	0	091	[
006	CTRL-F	049	1	092	\
007	CTRL-G	050	2	093	]
008	BS	051	3	094	↑
009	HOR. TAB	052	4	095	—
010	LINE FEED	053	5	096	`
011	VERT. TAB	054	6	097	a
012	FF	055	7	098	b
013	CR	056	8	099	c
014	CTRL-N	057	9	100	d
015	CTRL-O	058	:	101	e
016	CTRL-P	059	;	102	f
017	CTRL-Q	060	<	103	g
018	CTRL-R	061	=	104	h
019	CTRL-S	062	>	105	i
020	CTRL-T	063	?	106	j
021	CTRL-U	064	@	107	k
022	CTRL-V	065	A	108	l
023	CTRL-W	066	B	109	m
024	CTRL-X	067	C	110	n
025	CTRL-Y	068	D	111	o
026	CTRL-Z	069	E	112	p
027	CTRL-[	070	F	113	q
028	CTRL-\	071	G	114	r
029	CTRL-]	072	H	115	s
030	CTRL-↑	073	I	116	t
031	CTRL-—	074	J	117	u
032	SPACE	075	K	118	v
033	!	076	L	119	w
034	"	077	M	120	x
035	#	078	N	121	y
036	$	079	O	122	z
037	%	080	P	123	{
038	&	081	Q	124	|
039	'	082	R	125	}
040	(	083	S	126	~
041	)	084	T	127	DEL
042	*	085	U		

CTRL = Control Character BS = Backspace
CR = Carriage Return FF = Form Feed DEL = Rubout

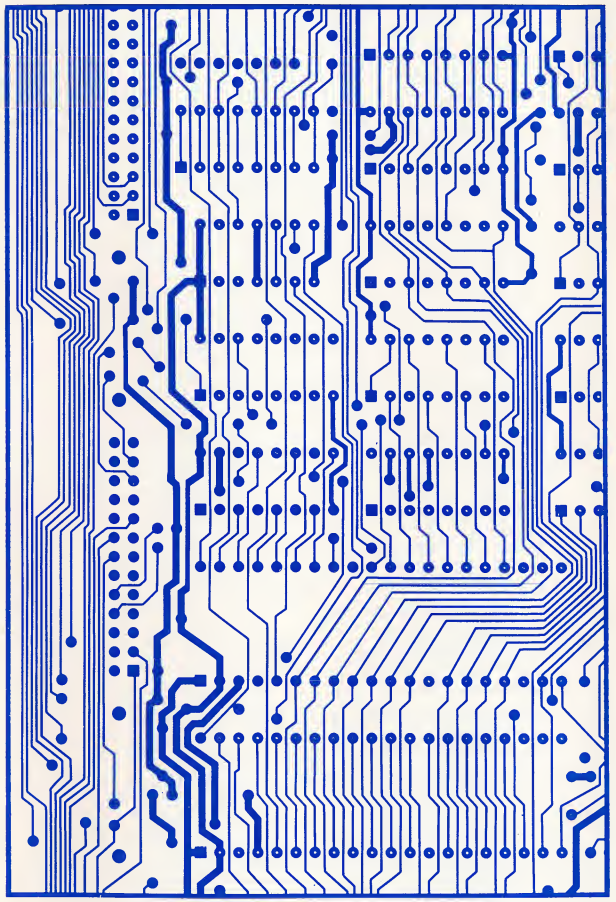
Cromemco 16K Basic Features

- Semi-compiling design — combines the best features of both interpreters and compilers — yields exceptionally fast execution times
- Allows 3 types of variables:
  - * integer (2 bytes) range +32767 to −32768
  - * short floating point (4 bytes) range ±9.99E+62 to ±9.99E−65
  - * long floating point (8 bytes) range ±9.99E+62 to ±9.99E−65
- 14 digit accuracy
- Advanced floppy disk I/O capabilities
- Binary and ASCII storage for both programs and data
- Sequential and random access files
- English language error messages
- Syntax error checking as program is entered
- Dynamic error trapping
- TRACE and immediate mode to facilitate debugging
- Advanced string handling capabilities
- Advanced output formatting capabilities
- Chaining of programs
- Direct machine language interaction

Cromemco incorporated
Specialists in computers and peripherals
280 Bernardo Ave., Mountain View, CA 94041

July 1978

Cromemco 16K Extended Basic Instruction Set



YOUR LOCAL DEALER IS

Cromemco 16K Basic Instructions

Abbreviations:

A, B, C, D	variables
m, n, p, r	integers
E, F, G, H	expressions or variables (H may be relational)
L1, L2	line numbers
stmnt	statement
strng	string expression
[ ]	optional parameter
{ }	choose one parameter
. . .	may be repeated
fn	file number
p1, p2	parameters
s	one letter
*	do not use with line no.
**	use only with line no.
†	disk basic only
byte	byte value
fmt	format
dr:	drive:

ABSolute value (E)
ASCii (A$)
ATN(E) arctangent
* **AUTOL**ine L1, L2

BINAND (A, B)
BINOR (A, B)
BINXOR (A, B)
† **BYE**

CHR$ (A) ASCI I character
CLOSE [\fn\]
* **CON**tinue
COSine (E)
† **CREATE** strng

** **DATA** [A] [strng] [,b . . .]
DEGree
DELETE L1, L2
DEF FNs (A) = E
DIM A (m) [, B(n, p, r) . . .]
†* **DIR**ectory [dr:] [strng]
† **DiSK** [dr:]

ECHO
** **END**
ENTER strng
† **ERASE** strng
ESCape
EXPonent (E)

{**FOR** A = E **TO** F [**STEP** G]
{**NEXT** A
FRActional part (E)
FREe space (E)

GET \fn [,p1 [,p2]] \ [E, F . . .]
GOTO L1
{**GOSUB** L1
{**RETURN**

IF H **THEN** {L1} {stmnt}
IMODE
INPUT [{\fn, p1, p2\} {strng}] A[,B . . .] [;]
INTEGER A [(m)] [, B . . .]
INTeger (E)
INP (m)
IOSTAT (fn, m)
IRN (E) integer random number generator

LENgth (A$)
[**LET**] A = E
LFMODE
LIST [strng,] [L1[,L2]]
LONG A [(m)]],B . . .]
LOGarithm (E)
† **LOAD** strng

MAXimum (E)
† **MAT** A = E
MINimum (E)

NOECHO
NOESCape
NoTRACE

ON E {GOTO} {GOSUB} L1
ON ERROR {STOP} {GOTO} {GOSUB} L1
ON ESCape {STOP} {GOTO} {GOSUB} L1
OPEN \fn [,p1[, p2]] \ strng
OUT m, byte

PEEK (m)
POSition (A$, Y$, n)
POKE m, byte
{**PRINT**} {@} [\fn, p1, p2\ ] [**USING** fmt] [E [{,} {;} F . . .]]
PUT \fn [,p1 [,p2]] \[E, F . . .]

RADians
RANDOMIZE
READ A [,B . . .]
REM [anything you want]
RESTORE
* **RENUMBER**
† **RENAME** strng-old, strng-new

RND (E) random number generator values
RUN [strng]

† **SAVE** strng
SCRatch
SET m,A
SFMODE
SGN (E) algebraic sign
SHORT A [(m)] [,B . . .]
SINe (E)
SPaCe (E) use with PRINT
SQR (E) square root
** **STOP**
STR$ (n) string
SYStem (E)

TANgent (E)
TAB (E) use with PRINT
TRACE

USeR (A, p1 [,p2 . . .])

VALue (A$)

Hexadecimal — Decimal Conversion Table

HEXADECIMAL COLUMNS							
4		3		2		1	
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	2	32	2	2
3	12,288	3	768	3	48	3	3
4	16,384	4	1,024	4	64	4	4
5	20,480	5	1,280	5	80	5	5
6	24,576	6	1,536	6	96	6	6
7	28,672	7	1,792	7	112	7	7
8	32,768	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,056	B	2,816	B	176	B	11
C	49,152	C	3,072	C	192	C	12
D	53,248	D	3,328	D	208	D	13
E	57,344	E	3,584	E	224	E	14
F	61,440	F	3,840	F	240	F	15
7 6 5 4		3 2 1 0		7 6 5 4		3 2 1 0	
BYTE				BYTE			